Design and Analysis Algorithms

تصميم و تحليل خوارزميات

ITGS301

Lecture 5 : المحاضرة الخامسة

Tripoli University - Faculty of Information Technology
Marwa Solla
m.solla@uot.edu.ly

$$\begin{tikzcd}
 & \text{Algorithm} \arrow[dl] \arrow[dr] & \\
\text{Iteration} \arrow[d] & & \text{Recursion} \arrow[d] \\
\begin{array}{c}T(n) = \text{sum of computing} \\ \text{time of all statements}\end{array} \arrow[d] & & T(n) = \text{recurrence equation} \arrow[d] \\
\text{Closed-end equation} \arrow[dr] & & \text{Closed-end equation} \arrow[dl] \\
 & \text{Classify Order} &
\end{tikzcd}$$

What is a Recursion ?

Recurrence Relations

When an algorithm contains a recursion call to itself, we can often describe the running time by ***recurrence equation*** ***or recurrence***. The recurrence describes the over all running time on the problem of size ***n*** in terms of the running time on smaller inputs. *Recurrence* is an equation that describes a function in term of its value on small inputs

A recurrence is an equation that is used to represent the running time of a recursive algorithm

Recurrence relations result naturally from the analysis of recursive algorithms, solving recurrence relations yields a *closed-end formula for calculation of run time.*

العلاقة التكرارية هي معادلة رياضية تستخدم لتمثل وقت الخوارزميات ذاتية الاستدعاء

Cases of a Recurrence Relations

A recursive algorithm has two cases:

(1) Base Case

(2) Recursive Case

General form of a Recurrence Relations

$$T(n) = \begin{cases} c & n \le 1 \quad \longleftarrow \text{Base Case} \\ aT(n/b) + f(n) & n > 1 \quad \longleftarrow \text{Recursive Case} \end{cases}$$

a: the number of times a function calls itself

b: the factor by which the input size is reduced

$f(n)$: the run time of each recursive call

For examples,

Example 1: the recursive Algorithm to compute n! :

```
/* Returns n!= 1*2*3...(n-1)*n for n >= 0. */
int factorial (int n)
 {
   if (n == 1) return 1;
   else
  return factorial (n-1) * n;
}
```

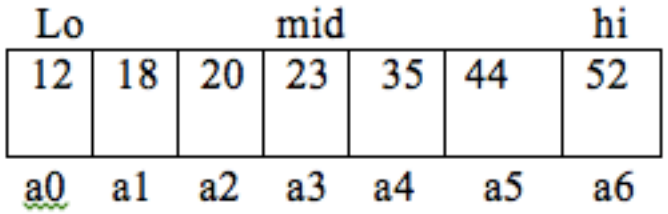
The running time, T(n), can be defined as recurrence equation:

$$T(n) = 1 \quad n=1$$
$$T(n) = T(n-1) + 1 \text{ for all } n>0$$

Example 2: .ΆyΠ·Ŋè {ŜĿ·ŊŎK /ƻŘŜ Άƴ /Σ ŊŜŎdzŊăΆØŜ Άƴ .fĿ·Ŏˌ ΆúŜŊĿ·úΆØŜ ΆƴDŊŜè

```
int binarySearch(int arr[], int low, int high, int x) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x)
            return mid;
        else if (arr[mid] < x)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

```
int binarySearch(int arr[], int l, int r, int x)
{
    // checking if there are elements in the subarray
    if (r >= l) {

        // calculating mid point
        int mid = l + (r - l) / 2;

        // If the element is present at the middle itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then it can only
        // be present in left subarray
        if (arr[mid] > x) {
            return binarySearch(arr, l, mid - 1, x);
        }

        // Else the element can only be present in right
        // subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not present in array
    return -1;
}
```



The running time, T(n), can be defined as recurrence equation:

$$T(n) = 1 \qquad n=1$$
$$T(n) = T(n/2) + 1 \text{ for all } n > 1$$

Exercise

```
1) int add (int x)
   {
      if (x == 1) return 5;
    else
      return 1 + add (n-1);
   }
```

Recurrence equation is:

$T(n) = 1 \quad n=1$

$T(n) = T(n-1) + 1$ for all $n>1$

Exercise

```
2) int power ( x , n)
   {
     if (n == 0)
        return 1;
     else
       if (n == 1)
          return x;
       else
          if (n % 2=0)
             return power(x,n/2) * power(x,n/2) ;
          else
             return return x *power(x,n/2) * power(x,n/2) ;
   }
```

Recurrence equation is:

$T(n) = 1 \qquad n=0$

$T(n) = 2 \qquad n=1$

$T(n) = 2T(n/2) + 1$ for all $n>1$

Solving Recurrence Relations

There are many methods to solve the recurrence relations, some of them are:

- Iteration method.
- The Master method.
- Recursion tree method.

ITERATION METHOD

Iteration method

Iteration is simply the repetition of processing steps. It is used to computing the running time for any recursive algorithm.

Note: We need to solve the recurrence equation by getting the Closed End formula, then calculation of running time.

We will show how this method works by some examples:

Example 1 *(*Factorial)

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & \text{for all } n>0 \end{cases}$$

Answer: Iteration T(n)

1 . $T(n) = T(n-1) + 1$

2 Since, $T(n-1) = T(n-1-1) + 1$

$= T(n-2) + 1$

then, $T(n) = T(n-2)+1+1$
$= T(n-2) + 2$

3 Since, $T(n-2) = T(n-2-1) + 1$
$= T(n-3) + 1$

then, $T(n) = T(n-3) + 1 + 2$
$= T(n-3) + 3$

4 Since, $T(n-3) = T(n-3-1) + 1$
$= T(n-4) + 1$

then, $T(n) = T(n-4) + 1 + 3$
$= T(n-4) + 4$

n $\qquad T(n) = T(n-n) + n$

$$= T(0) + n$$

$$= 1 + n$$

The closed end formula: $T(n) = 1 + n$

the running time $T(n) = O(n)$

*Example 2 (*Binary Search)

Find the closed end formula using the iteration method.

$$T(n) = \begin{cases} 1 & n=1 \\ T(n/2) + 1 & \text{for all } n > 1 \end{cases}$$

answer

1 $T(n) = T(n/2) + 1$

2 Since, $T(n/2) = T(n/4) + 1$

Then, $T(n) = T(n/4) + 1 + 1$

$= T(n/4) + 2$

3 Since, $T(n/4) = T(n/8) + 1$

$T(n/2^2) = T(n/2^3) + 1$

Then, $T(n) = T(n/2^3) + 1+2$

$= T(n/2^3) + 3$

.

.

n $T(n) = T(n/2^k) + k$

Since $T(n) = 1$ suppose that $n/2^k$

$n = 2^k$ $k=\log_2 n$ $k= \lg n$

$T(n) = T(1) + k$

$= T(1) + \lg n$

The closed end formula = 1 +lg n

The running time T(n) is O(lg n).

Example 3:

$$T(n) = \begin{cases} 0 & n=0 \\ 2T(n-1) + 1 & \text{for all } n > 0 \end{cases}$$

answer

1. T(n) = 2T(n-1) + 1

2 $T(n-1) = 2T(n-2) + 1$

Then $T(n)= 2[2T(n-2) + 1] +1$

$= 4T(n-2)+ 2+1$

3 $T(n-2) = 2T(n-3) + 1$

Then $T(n) = 4[2T(n-3) + 1] +2+1$

$= 8T(n-3) + 4+ 2+ 1$

$= 2^3 T(n-3) + 2^2 + 2+ 1$

.

.

n $T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + ... + 2^1 + 2^0$

When n=0

$n - k = 0 \rightarrow k = n$

$T(n) = 2^n\, T(n-n) + 2^{n-1} + 2^{n-2} + \ldots. + 2^1 + 2^0$

$= 2^n \,.\, T(0) + ⍰^{n-1}\; 2^k$

$= 2^n.0 + [2^{n-1+1} - 1/2-1]$

$= 2^n.0 \quad + [2^n \;-1]$

$= 2^n \;-1$

The closed end formula = $2^n \;-1$

The running time = $O(2^n)$

The End .